React - Komponentin elinkaari ja AJAX

> Tämä kappale ei ole erityisen pitkä, mutta se sisältää kaksi esimerkkiä (react10-99.html ja react11-99.html), joiden ymmärtäminen saattaa vaatia Reactin perusteisiin juuri tutustuneelta aikaa. Panostuksesi palkitaan jatkossa, joten ole kärsivällinen. Jos resurssisi on kortilla, silmäile ensimmäisen esimerkin lähdekoodi kursorisesti läpi ja panosta täysillä ainakin jälkimmäiseen.

React-komponentin elinkaari

React-komponentilla on kolme päätilaa sen elinkaaren aikana:

- alustus (mounting to DOM),
- päivittäminen (render) ja
- poistaminen (unmounting DOM)

Toiminnoista:

- `will`-määreisiä kutsutaan juuri ennen kuin ko. asia tapahtuu komponentille ja
- `did`-määreisiä tapahtuman jälkeen.

Täydellinen lista elinkaaren metodeista: The Component Lifecycle

Komponentin alustusvaihe

Komponentin luonnin yhteydessä ja sitä DOM-puuhun liitettäessä suoritetaan seuraavat metodit:

Method	Description
getDefaultProps	*) oletusarvojen asettaminen this.props-objektiin
getInitialState	**) oletusarvojen asettaminen this.state-objektiin
componentWillMount	kutsutaan kerran ennen ensimmäistä renderöintiä
render	komponentin piirtäminen näytölle
componentDidMount	kutsutaan kerran ensimmäisen renderöinnin jälkeen, DOM-puu ...

```
componentDidMount     kutsutaan kerran ensimmäisen renderöinnin jälkeen, DOM-puu v
```

*) ES6-syntaksilla defaultProps asetetaan luokan määrittelyn jälkeen seuraavasti

```
class Person extends React.Component {
  // ...
}

Person.defaultProps = {
  name: 'Alainen Kim'
};
```

**) ES6-syntaksilla tilamuuttujan alustus asetaan luokan constructor-metodissa

```
class Person extends React.Component {
  constructor(props) {
    super(props);
    this.state = {name: 'Alainen Kim'};
  }
```

Komponentin päivitys

Komponentin ollessa DOM-puussa, se voi päivittää tietojaan state- ja props-objektien avulla:

```
Method                                          Description

componentWillReceiveProps(nextProps)            komponentti saa uusia props-arvoja

shouldComponentUpdate(nextProps, nextState)     palauta true -> render, palauta false

render                                          komponentin piirtäminen näytölle

componentDidUpdate(prevProps, prevState)        kutsutaan seuraavien renderöintikerto
```

> Reactin yhteydessä puhutaan ns. Virtual DOM -käsitteestä. Tämä tarkoittaa sitä, että React pitää yllä "omaa" DOM-puuta, josta render-toiminto päivittää selaimen oikeaan DOM-puuhun vain muuttuneet sisällöt. Tämä on näkyvissä hyvin seuraavassa esimerkissä: Hello World in React. Avaa esimerkki selaimeen ja katso selaimen inspectorin kautta, kuinka vain kellon aika päivittyy DOM-puuhun elements-näkymässä.

Komponentin poistaminen

Komponentin poistaminen DOM-puusta aiheuttaa seuraavan metodikutsun:

Method	Description
componentWillUnmount	komponentti poistetaan DOMista

Esimerkki: react10-99.html

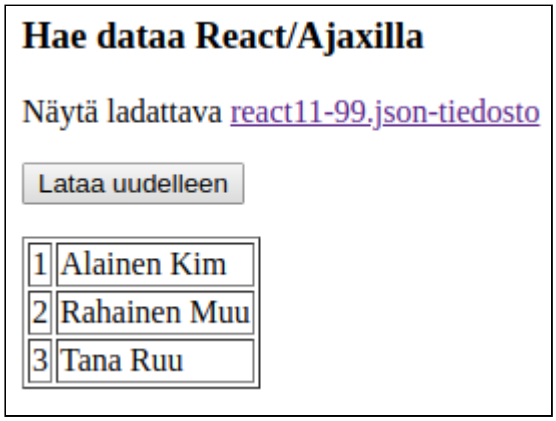
Kokeile, muuta ja ymmärrä: react10-99.html-esimerkissä esitellään React-komponenttien elinkaarta kahden komponentilla avulla. Tähän asti opintojaksolla mukana sinnitelleellä opiskelijalla kuluu esimerkin sisäistämiseen aikaa osaamistasosta riippuen noin 10-80min.

React ja ulkoisten tietolähteiden käyttäminen

- Reactissa ei sisäänrakennettuna keinoja: Käytetään esim. jQueryn tarjoamaa Ajax-toiminnallisuutta
- Pienissä sovelluksissa ulkoinen data voidaan ladata suoraan komponenttiin ja käyttää sitä `this.state`-objektin kautta
- Isommissa toteutuksissa data kannattaa ladata sovelluksen juuri-komponenttiin ja jaella sieltä tarvittaessa `this.props`-objektin kautta edelleen
- Tarvittaessa dataa voidaan ladata myös erillisiin komponentteihin ja edelleenjakaa sieltä.
- Jos data pitää ladata sivun lataamisen yhteydessä ilman käyttäjän toimenpiteitä, se suositellaan tehtäväksi komponentin `componentDidMount`-metodissa, tällöin DOM-puu on valmis ja ladattu data voidaan esittää käyttöliittymässä.

Ajax-toiminnallisuus komponentissa - react11-99.html

Toteutetaan pieni React-sovellus, jossa käyttäjä voi Näytä henkilöt -painiketta klikkaamalla hakea JSON-tiedostosta dataa näytettäväksi käyttöliittymään. JSON-tiedostossa mahdollisesti muuttunut data voidaan myös ladata uudelleen:



JSON-tiedoston sisältö on yllä nähdylle listaukselle seuraava:

```
{
    "persons":[
      { "id":1, "name":"Alainen Kim"},
      { "id":2, "name":"Rahainen Muu"},
      { "id":3, "name":"Tana Ruu" }
    ]
}
```

ja sovelluksen HTML-tiedosto seuraavanlainen. Huomaa, että tämä ratkaisu käyttää jQUery-kirastoa, joten se on ladattu käyttöön rivillä 18.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8" />
    <title>react11-99.html</title>
</head>
<body>

   <h3>Hae dataa React/Ajaxilla</h3>
   <p>Näytä ladattava <a href="react11-99.json">react11-
    <div id="root"></div>

    <script src="https://unpkg.com/react@16/umd/react.de
```

```
14      <script src="https://unpkg.com/react-dom@16/umd/reac
15      <script src="https://unpkg.com/babel-standalone@6.15
16
17
18      <script src="https://code.jquery.com/jquery-3.3.1.mi
19
20      <script type="text/babel" src="react11-99.js"></scri
21      </body>
22  </html>
```

Huomaa seuraavat asiat varsinaisesta React-osuuden lähdekoodista

- Rivi 5: Tilamuuttuja persons sisältää AJAXilla ladatun datan taulukossa, loaded pitää yllä tietoa siitä, onko JSON-tiedoston dataa saatu vielä ladattua. Nämä ovat oletusarvot: tyhjä taulu ja false
- Rivit 12-18: Aluksi kun dataa ei ole vielä ladattu, näytetään käyttäjälle painiketta lataamisen suorittamiseksi.
- Rivit 37-40: Kun käyttäjä on klikannut kumpaa tahansa latauspainikkeista, tilamuuttujat asetetaan alkuarvoihinsa ja data haetaan komponentin getPersons-metodilla
- Rivit: 43-53: Data haetaan jQueryn ajax-toiminnallisuudella
- Rivit 48-49: JSON-tiedostosta haettu data on data-muuttujassa objektina, jonka ainoa persons-kenttä sisältää 3 henkilöobjektia taulussa. Tämä sisältö kopioidaan lamda-merkittyllä (nuolifunktio) funktiolla tilamuuttujaan this.state.persons. Samalla loaded asetaan arvoon true, jotta data tulee näytetyksi käyttöliittymässä (render-metodilla). Lambdaa käyttämällä rivin 49 this-viittaus viittaa nimenomaan Persons-komponenttiin (ei jQueryn Ajax-objektiin). Kommentoidut kaksi muuta vaihtoehtoa datan hakemiseen näyttävät, millaisiin ratkaisuihin jouduttaisiin ilman Lambdaa.
- Riveillä 25-28 renderöidäään tilamuuttujan persons-taulukon objektit käyttöliittymään

```
1   // Highscores component
2   class Persons extends React.Component {
3       constructor(props) {
4         super(props);
5         this.state = {persons: [], loaded:false};
6         this.onLoadClick = this.onLoadClick.bind(this);
7         this.getPersons = this.getPersons.bind(this);
8       }
9
10      render() {
```

```
10      render() {
11          // Jos dataa EI ole VIELÄ ladattu:
12          if (!this.state.loaded) {
13              return (
14                  <div>
15                      <button onClick={this.onLoadClick}>N
16                  </div>
17              );
18          }
19          // Näytetään haettu JSON-data:
20          return (
21              <div>
22                  <p><button onClick={this.onLoadClick}>La
23                  <table border="1"><tbody>
24                  {
25                      this.state.persons.map((person, inde
26                              <tr key={index}>
27                              <td>{person.id}</td><td>{per
28                              </tr>
29                      )
30                  }
31                  </tbody></table>
32              </div>
33          );
34      }
35
36      // Tyhjätään mahdollisesti aiemmin haettu data ja ha
37      onLoadClick () {
38          this.setState({persons:[], loaded:false});
39          this.getPersons();
40      }
41
42      // ladataan henkilötiedot TAPA1 (siistein)
43      getPersons () {
44          $.ajax({
45              url: 'react11-99.json',
46              cache: false,
47              dataType: 'json'
48          }).done((data) => {
49              this.setState({persons: data.persons, loaded
```

```
50          }).fail((jqXHR, textStatus, errorThrown) => {
51              console.log(textStatus+":"+errorThrown);
52          });
53      }
54
55  /*
56      // ladataan henkilötiedot TAPA2
57      // https://stackoverflow.com/questions/39564938/gett
58      getPersons = () => {
59          $.ajax({
60              url: 'react11-99.json',
61              cache: false,
62              dataType: 'json'
63          }).done(function(data) {
64              this.setState({persons: data.persons, loaded
65          }.bind(this)).fail(function(jqXHR, textStatus, e
66              console.log(textStatus+":"+errorThrown);
67          });
68      }
69  */
70
71  /*
72      // ladataan henkilötiedot TAPA3 (vanha)
73     // https://stackoverflow.com/questions/39564938/getti
74
75      getPersons () {
76          var me = this;
77          $.ajax({
78              url: 'react11-99.json',
79              cache: false,
80              dataType: 'json'
81          }).done(function(data) {
82              me.setState({persons: data.persons, loaded:t
83          }).fail(function(jqXHR, textStatus, errorThrown)
84            console.log(textStatus+":"+errorThrown);
85          });
86      }
87      */
88  }
89
```

```
// render component
ReactDOM.render(
    <Persons />,
    document.getElementById("root")
);
```

Lisäesimerkkejä

Oheiset esimerkit valaisevat osin asioita, joita yllä ei ole esitetty. Osin ohjelmakoodit on kirjoitettu hieman vanhemmalla Reactin syntaksilla. Ohjelmakoodeja tutkimalla on kuitenkin mahdollista kirkastaa ja syventää tietämystä Reactin perusteista.

Kokeile, muuta ja yritä ymmärtää seuraavista ainakin esimerkit 10-12

- React-esimerkit 7-12

Jätetty tarkoituksella tyhjäksi